Carleton University, Technical Report SCE-15-04 August 2015

# *UML diagram synthesis techniques: a systematic mapping study*

Damiano Torre[1,2], Yvan Labiche[1], Marcela Genero[2]

[1]Carleton University, Department of Systems and Computer Engineering, Software Quality Engineering Laboratory
1125 Colonel By Drive, Ottawa, ON K1S5B6, Canada
{dctorre,labiche}@sce.carleton.ca

[2]University of Castilla-La Mancha, Department of Technologies and Information Systems, ALARCOS Research Group,
Paseo de la Universidad, 4 13071 Ciudad Real, Spain
marcela.genero@uclm.es

**Abstract.**
**Context:** The Unified Modeling Language (UML), with its 14 different diagram types, is the de-facto standard modeling language for object-oriented modeling and documentation. Since the various UML diagrams describe different aspects of one, and only one, software under development, they are not independent but strongly depend on each other in many ways. In other words, diagrams must remain consistent. Dependencies between diagrams can become so intricate that it is sometimes even possible to synthesize one diagram on the basis of others. Support for synthesizing one UML diagram from other diagrams can provide the designer with significant help, thus speeding up the design process, decreasing the risk of errors, and guaranteeing consistency among the diagrams.
**Objective:** The aim of this article is to provide a comprehensive summary of UML synthesis techniques as they have been described in literature to date in order to obtain an extensive and detailed overview of the current research in this area.
**Method:** We have performed a Systematic Mapping Study by following well-known guidelines. We selected ten primary studies by means of a search with seven search engines performed on October 2, 2013.
**Results:** Various results are worth mentioning. First it appears that researchers have not frequently published papers concerning UML synthesis techniques since 2004 (with the exception of two papers published in 2010). Only half of the UML diagram types are involved in the synthesis techniques we discovered. The UML diagram type most frequently used as the source for synthesizing another diagram is the sequence diagram (66.7%), and the most synthesized diagrams are the state machine diagram (58.3%) and the class diagram (25%).
**Conclusion:** The fact that we did not obtain a large number of primary studies over a 14 year period (only ten papers) indicates that synthesizing a UML diagram from other UML diagrams is not a particularly active line of research. Research on UML diagram synthesis is nevertheless relevant since synthesis techniques rely on or enforce diagram consistency, and studying UML diagram consistency is an active line of research. Another result is that research is needed to investigate synthesis techniques for other types of UML diagrams than those involved in our primary studies.

**Keywords:** Unified Modeling Language (UML), UML synthesis techniques, UML consistency, Systematic Mapping Study

## 1 INTRODUCTION

The Model Driven Architecture (MDA) [1] promotes a set of transformations between successive models from requirements to analysis, to design, to implementation, and to deployment [2]. Much attention has been paid to MDA by academia and industry in recent years [3], which has resulted in the models gaining even more importance in software development. The Unified Modeling Language (UML) [4] is the specification most frequently used by the Object Management Group (OMG) and is the de-facto standard tool for object-oriented modeling and documentation [5]. It is the privileged modeling tool when implementing the MDA. The architecture of the UML is based on a four-layer meta-model structure, and it provides 14 diagram types [4] that can be used to describe a system from different perspectives (e.g., structure, behavior) or abstraction levels (e.g., analysis, design), which helps deal with complex systems and distribute responsibilities among stakeholders, among other benefits. Since the various UML diagrams describe different aspects of one, and only one, software under development, they are not independent but strongly depend on each other in many ways. In other words, the diagrams must be consistent [6]. Dependencies between diagrams can become so intricate that it is sometimes even possible to synthesize one diagram from other diagrams [7, 8].

The term "synthesis" has traditionally been used to represent the automatic construction of a program or a behavioral model from formal requirements [9]. In this paper, the accepted basic definition of synthesis is that of generating a UML diagram from one of the 14 types on the basis of already existing diagrams of other types [7, 8]. Support for synthesizing one UML diagram from other diagrams can provide the designer with significant help, thus speeding up the design process, decreasing the risk of errors, and guaranteeing consistency among the models [10].

Although many researchers have proposed, explicitly or otherwise, techniques with which to synthesize different types of UML diagrams, no well-accepted and as complete as possible set of UML synthesis techniques has been described to date. Our main research question is therefore: What is the current state-of-the-art in UML diagram synthesis?

This endeavor additionally fits into another research activity we are conducting, which focuses on the consistency between UML diagrams that describe one software system [6]. Indeed, while synthesizing one diagram from others, a synthesis technique enforces some consistency between the source diagram(s) and the generated diagram. However, synthesis techniques may or may not be explicitly rooted in a set of consistency rules. A further objective of this paper is therefore to discover the consistency rules that result from UML diagram synthesis techniques.

To achieve these goals, we have performed a Systematic Mapping Study (SMS) [11], following the guidelines of Kitchenham and Charters [12], as this is a research method that provides an objective procedure with which to identify the quantity of existing research related to a research question. Performing an SMS has several benefits [13]: it provides a starting point for PhD students, and in the longer term it provides a body of knowledge for the next generation of researchers.

A description of the SMS protocol followed [12] is shown as follows (Section 2). We then discuss the results (Section 3). The paper ends with a discussion of the threats to validity (Section 4) and our conclusions (Section 5).

## 2 Systematic Mapping Study—PLANNING and EXECUTION

In this section we present the main components of the planning of our SMS [12], which are: the specification of research questions the study should aim to answer (section 2.1); the strategy followed to search for existing research (section 2.2); the procedure followed to select (or reject) papers for further investigation (section 2.3); and the procedure used to extract data from selected papers in order to answer the research questions (section 2.4). We then discuss the execution of the plan (section 2.5).

### *2.1* Research questions

The underlying motivation for the research questions was to determine the current state-of-the-art as regards UML synthesis techniques, and the underlying consistency rules. We therefore considered five research questions (RQs): Table 1.

**Table 1.** Research questions

| Research questions | Main motivation |
|---|---|
| **RQ1:** Which UML versions are used by researchers in the synthesis techniques found? | To discover which UML versions are used in the approaches that handle the UML synthesis techniques and whether there are any differences between UML versions (possibly owing to differences between metamodel versions). |
| **RQ2**: Which types of UML diagrams have been used as the source and target of synthesis techniques? | To discover the UML diagrams that research has focused upon in order to reveal the UML diagrams that are most frequently considered in this topic, and perhaps those that lack attention. |
| **RQ3:** Which technologies are used to implement synthesis techniques? | To find the technologies that are used to implement UML diagram synthesis techniques. From the outset it is possible to think of, for instance, model transformation technologies and ad-hoc transformation algorithms. |
| **RQ4:** Are the synthesis techniques automatic or manual? | To discover whether the UML diagram synthesis techniques are automated or require human input. |
| **RQ5:** What types of research papers report on UML synthesis techniques? | To determine whether the field is generally more applied or consists of more basic research, as evidenced when classifying papers according to a well known taxonomy [14]. |

### 2.2 Search strategy

Conducting a search for research papers requires the identification of a search string, and the specification of the parts of research papers in which the search string will be sought (the search fields). We identified our search string by following the procedure of Brereton et al [15], which is composed of five steps: (1) Define the major terms; (2) Identify alternative spellings, synonyms or related terms for major terms; (3) Check the keywords in any relevant paper that may already be available to consolidate the list of terms; (4) Use the Boolean OR to incorporate alternative spellings, synonyms or related terms; (5) Use the Boolean AND to link the major terms.

In our case, the major search terms were "UML" and "Synthesis" and the alternative spellings, synonyms or terms related to the major terms were:

- UML ::= (uml OR unified modeling language OR unified modelling language)
- Synthesis ::= synthesis diagram

When selecting the search string, we considered various alternatives with the following objective in mind: we were interested in collecting synthesis techniques, or in other words, in identifying synthesis algorithms in order to precisely understand them and identify the consistency rules they (implicitly) enforce. Other search strings were experimented with, but cannot be discussed in this paper owing to space limitations. Of all the alternative search strings in the set, we selected the following as it allowed us to retrieve the largest number of useful papers, i.e., the largest number of papers focusing on UML diagram synthesis:

*((uml OR unified modeling language OR unified modelling language) AND (Synthesis Diagram))*

The search was limited to electronic papers and considered only peer-reviewed journals, international conferences and workshops in only the English language. We did not establish any restriction as regards publication years, except that we stopped with papers published in October 2013 at the latest. We used the above mentioned search string with the following seven search engines, as has occurred in other works [12]: Engineering Village, Google Scholar, CiteSeer, IEEE Digital Library, Science Direct, ACM Digital Library, and Inspec database. The searches were limited to the following search fields: title, keywords and abstract.

### 2.3 Selection procedure and inclusion and exclusion criteria

Since a systematic, automated search in databases based on a search string typically returns papers that cannot be used to answer a set of research questions of interest in an SMS (e.g., in our case, a paper that mentions the synthesis of test cases from a UML sequence diagram), the set of papers collected from the search needs to be trimmed in a systematic manner. This is typically done by relying on the so-called inclusion and exclusion criteria. Inclusion and exclusion criteria are based on the research questions and are piloted to ensure that they can be reliably interpreted and that they classify studies correctly [12]. In this section we discuss the inclusion and exclusion criteria used. We then discuss the process followed to include research papers in the mapping study. According to standard terminology in empirical software engi-

neering, the set of research papers finally retained is referred to as the primary studies [12].

The inclusion criteria were: (i) Electronic papers focusing on UML diagram synthesis which contained at least one UML synthesis technique; (ii) Electronic papers written in English; (iii) Electronic papers published in peer-reviewed journals, international conferences and workshops; (iv) Electronic papers published until October 2, 2013; (v) Electronic papers which proposed UML synthesis techniques.

The exclusion criteria were: (i) Electronic papers not focusing on UML diagram synthesis; (ii) Electronic papers which were extended abstracts; (iii) Duplicated electronic papers (e.g., returned by different search engines); (iv) Electronic papers which discussed synthesis techniques between UML diagrams and other non-UML sources of data, such as requirements or source code (i.e., reverse engineering).

### 2.4 Data extraction strategy

We extracted the data from the primary studies according to a number of criteria, which were directly derived from the research questions detailed in Table 1. Using each data extraction criterion required that we read the full-text of each primary study. Once extracted, we recorded data on an Excel spreadsheet that represented our data form. The following information was collected from each primary study:

- Search engines: where the paper was found (see section 2.2);
- Inclusion and Exclusion Criteria (see section 2.3);
- Data related to Research Questions (see Section 2.1):
  - Which UML version was used;
  - What the UML synthesis techniques are;
  - The UML diagrams that were involved as input and output in synthesis techniques: Class Diagram (CD), Collaboration Diagram (COD), Use Case Diagram (UCD), Communication Diagram (COMD), State Chart Diagram (SCD), Sequence Diagram (SD), Protocol State Machine Diagram (PSMD), Object Diagram (OD), Interaction Diagram (ID), Activity Diagram (AD), Composite Structure Diagram (CSD), Timing Diagram (TD), Interaction Overview Diagram (IOD), and Deployment Diagram (DD);
  - Tool support: Automatic signifies that the UML synthesis techniques were full-automated, i.e., supported by an implemented and working tool; Manual signifies that the UML synthesis techniques were not supported by any implemented and automatic tool.
  - Type of research used in the primary study, for which we used the following classification [14]: An *Evaluation Research* (ER) paper investigates techniques that are implemented in practice, and reports on their evaluation. Such a paper shows how the technique is implemented in practice (solution implementation) and what the consequences of the implementation are in terms of benefits and drawbacks (implementation evaluation). A *Proposal of Solution* (PS) paper proposes a solution to a problem and argues for its relevance, without a full-

blown validation. A *Validation Research* (VR) paper investigates the properties of a solution that has not yet been implemented in practice. A *Philosophical Paper* (PP) sketches a new way of looking at things, a new conceptual framework, etc. An *Opinion Paper* (OP) contains the author's opinion about what is wrong or good about something, how something should be done, etc. A *Personal Experience Paper* (PEP) places more emphasis on what than on why.

### 2.5 Execution

The planning for this SMS with the seven search engines began in July 2013 and was completed in October 2013. In this section we present the use of the search string with these engines and the selection of primary studies according to the inclusion/exclusion criteria previously described. In order to document the review process with sufficient details [12], we describe the multi-phase process of the four sub-phases employed:

- First sub-phase (SP1): the search string was used to search in the seven search engines, as mentioned earlier.
- Second sub-phase (SP2): we deleted duplicates.
- Third sub-phase (SP3): we obtained an initial set of studies by reading the title, abstract and keywords of all the papers obtained after SP2 while enforcing the inclusion and exclusion criteria. When this was not sufficient to decide whether or not to include or exclude a paper, we checked the full-text of the paper.
- Fourth sub-phase (SP4): the papers from SP3 were read in their entirety while reapplying the exclusion criteria. This resulted in the final set of primary studies.

Table 2 breaks down the number of papers we have found into sub-phases. We eventually collected ten primary studies for further analysis [7, 8, 10, 16-22]. These were then analyzed according to the criteria detailed in section 2.4 and data was recorded in an Excel file. Two papers [23, 24] found by the CiteSeer search engine were not included in the primary studies because they led to a subsequent, more complete publication [10] by the same authors. We only show results for five out of the seven search engines because Google Scholar did not return any results and Engineering Village and Inspec used the same search engine interface and database.

**Table 2.** Summary of primary study selection

| Sub phase | IEEE | Inspec | ACM | CiteSeer | Science Direct | Total |
|---|---|---|---|---|---|---|
| **SP1: Raw results** | 22 | 92 | 7 | 25 | 5 | **164** |
| **SP2: No duplicates** | 20 | 20 | 7 | 13 | 3 | **67** |
| **SP3: First selection** | 8 | 7 | 6 | 4 | 0 | **25** |
| **SP4: Primary studies** | 3 | 2 | 2 | 3 | 0 | **10** |

# 3 Systematic Mapping Study—RESULTS

A quantitative summary of the results for research questions RQ1-RQ5 is presented in Table 3. More details are provided in sections 3.1 to 3.5. We then discuss some of the main findings (section 3.6). (Percentages were rounded to the nearest integer value.)

## 3.1 UML version (RQ1)

Although UML 2.x has an improved semantics in comparison to UML 1.x, which could help devise diagram synthesis techniques, we did not find more synthesis techniques for UML 2.x (three papers, i.e., 30%) than for UML 1.x (7 papers, i.e., 70%) during the period 1999—2010 (period of publication of the ten primary studies): Table 3.

**Table 3.** Results of our SMS

| **Research question** | **Possible Answer** | | **Result** | |
|---|---|---|---|---|
| | | | # Papers | Percentage |
| RQ1: UML versions | UML 1.3 | | • 5 | • 50% |
| | UML 1.4 | | 2 | 20% |
| | UML 2.0 | | 1 | 10% |
| | UML 2.1. | | 1 | 10% |
| | UML 2.1.2 | | 1 | 10% |
| RQ2: UML diagrams | Diagram to start the synthesis | IOD | 1 | 8.3% |
| | | SCD | 2 | 16.7% |
| | | SD | 8 | 66.7% |
| | | COD | 1 | 8.3% |
| | Diagram to synthesize | CD | 3 | 25% |
| | | SCD | 7 | 58.3% |
| | | AD | 1 | 8.3% |
| | | SD | 1 | 8.3% |
| RQ23: UML Synthesis techniques | IOD → SCD | | 1 | 8.3% |
| | SD → SCD | | 5 | 41.7% |
| | SD → CD | | 2 | 16.7% |
| | SD → AD | | 1 | 8.3% |
| | COD → SCD | | 1 | 8.3% |
| | SCD → SD | | 1 | 8.3% |
| | SCD → CD | | 1 | 8.3% |
| RQ5: Research methods | Evaluation Research | | 1 | 10% |
| | Validation Research | | 1 | 10% |
| | Proposal of Solution | | 8 | 80% |
| RQ6: Support | Automatic | | 8 | 80% |
| | Manual | | 2 | 20% |

It is interesting to note that we found more papers on the synthesis of UML 1.x diagrams than on UML 2.x in a shorter period of time (four years in the case of UML 1.x versus seven years in the case of UML 2.x). No UML version was reported in one paper [21], but as it was published in 2003 which is the same year of publication as the UML version 2.0 [4], we classified it as a UML version 1.4 paper.

### 3.2 UML diagrams (RQ2)

Figure 1 summarizes the number of times each diagram (CD, IOD, COD, SCD, SD and AD) has been found to be the source or target of a synthesis. The sequence diagram (SD) is by far the diagram type that is most frequently used for the synthesis of a diagram of another type (66.7%, eight synthesis techniques). Conversely, the state chart diagram (SCD) is the most synthesized diagram type (58.3%, seven techniques), followed by the class diagram (CD) (25%, three techniques). Overall, only half of the UML 2 diagram types are involved in some sort of synthesis technique, and it is not entirely surprising that the diagram types involved in synthesis techniques are also identified as the most frequently used diagrams [25].

### 3.3 Technologies used for synthesis techniques (RQ3)

In the ten primary studies, we identified 12 different synthesis techniques, each of which involved one input diagram and one output diagram, which can be classified into seven different types of synthesis techniques: Figure 2 and **Error! Reference source not found.**. The two most frequently described synthesis technique types are sequence diagram to state chart diagram (SD→SCD) and sequence diagram to class diagram (SD→CD) for a total of nine techniques (58.3%). Each of the 12 synthesis techniques is summarized below, and we also discuss the technologies they rely on in order to perform the synthesis.

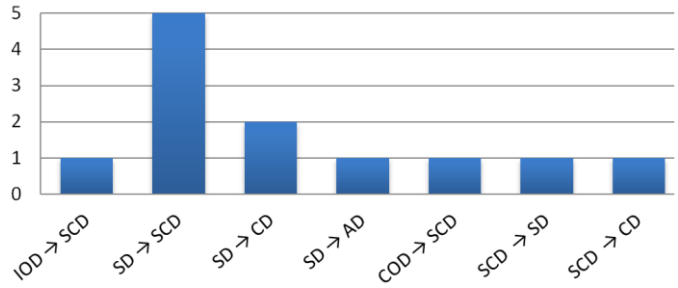


**Fig. 2.** From→To synthesis techniques

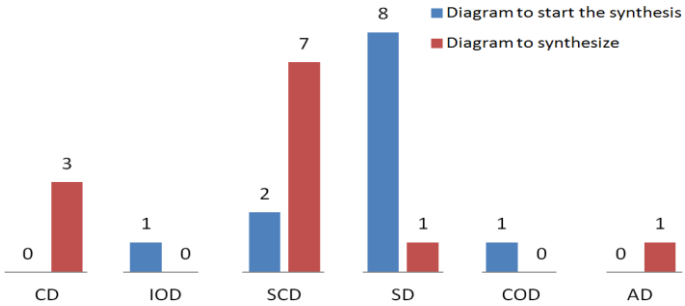


**Fig. 1.** UML diagrams in the synthesis techniques

Due to the space limit, in this section we only briefly cite the basic detail about the 12 UML synthesis techniques considered in this paper.

Whittle and Jayaraman [16] presented a technique to automatically synthesize hierarchical state machine for state dependent objects from Extended Interaction Overview Diagrams (EIODs), that is a three-layer structure, with a precise semantics, where the top layer defines use cases, the second layer defines scenarios, and the bottom layer defines sequence diagrams. We classify this synthesis technique as *precise mapping* since, even though the authors precisely describe the mapping between elements of the EIOD to elements of hierarchical state machines, they did not provide a detailed algorithm or did not rely on some other technology (e.g., model transformation) to perform the synthesis.

So far, as mentioned earlier, research mostly focused on technical issues such as how to best derive SCD from SD and how to best ensure the correctness of this synthesis: half (five) of the UML synthesis techniques address this issue. The Minimally Adequate Synthesizer (MAS) [10] infers a SCD from a set of SDs, using an engineer-guided grammatical inference technique: the state machine is seen as describing an unknown grammar to be inferred, of which the sequence diagrams are sentences. We classify this synthesis technique as *grammatical inference*. Ziadi and colleagues [18], very similarly to Whittle and Jayaraman [16], introduce an algebraic framework for such a synthesis: the authors define an algebraic representation of sequence diagrams, an algebraic representation of state machine diagrams, and then a precise mapping between the two. We classify this technique as *algorithmic* since the authors provide a detailed algorithm for the mapping. Schumann [19] synthesized state machines from sequence diagrams which messages are annotated with OCL pre and post conditions, based on an earlier work of the author [26], which is itself an earlier work of Whittle [16] that does not account for interaction fragments. We classify this technique as *algorithmic* since the earlier paper provides a detailed algorithm for the synthesis. The Fujaba project [21] includes such a synthesis too, which is based on grammatical inference as in MAS [10], and that we therefore also classify as *grammatical inference*. Selonen and colleagues also rely on a *grammatical inference* algorithm to synthesize a series of state machines from sequence diagrams [7].

Khriss and colleagues [20] proposed an incremental UML synthesis technique to generate from a set of CODs the SCDs of all the objects involved. The technique employed to perform the synthesis is informally described and we therefore classify this paper as *informal description*.

These different synthesis approaches [16] [10] [18] [19] [21] [7] [20] are very similar in the sense that they recognize object interactions, provided in various, though similar forms (Extended Interaction Overview Diagrams, sequence diagrams, collaboration diagrams) as specifying legal sequences of messages objects can receive and from which the complete set of legal sequences of messages objects can receive can be constructed (i.e., the state machine diagrams).

Selonen and colleagues presented the synthesis of UML CDs from SDs in 2000 [22], which they extended in 2001 [8] and summarized, along with other synthesis techniques (see below) in 2003 [7]. The authors informally discussed how elements

of sequence diagram can be mapped to elements of the class diagram. We therefore classify this technique as informal description.

Kang and colleagues [17] synthesized Ads from SDs thereby representing the flow of messages in sequence diagrams as activity diagrams. The mapping rules from SD to AD are precisely discussed and the overall synthesis is described with an algorithm. We therefore classify the technique as *algorithmic*.

We already discuss two of the synthesis techniques that Selonon and colleagues summarized [7]. They also discuss the synthesis of CD from SCD, as well as the synthesis of SCD from SD following the work of Systä [27]. These are only summaries, which we classify as *informal descriptions.*

### 3.4 Research papers (RQ4)

As reported in Table 3, the primary studies can be classified as ER (Evaluation Research), VR (Validation Research), and PS (Proposal of Solution), contributing 10%, 10% and 80% of the total, respectively. The vast majority of the primary studies therefore propose a complete technical solution for the synthesis of UML diagrams. One paper was categorized as an evaluation research paper [8] and one other was categorized as a validation research paper [17].

### 3.5 Type of support (RQ5)

As is described in Table 4, the synthesis techniques we collected are usually supported by a tool (80%, 8 papers). Only one of them [16] is integrated with its UCSIM UML CASE tool, a vendor-independent implementation that the authors intended to integrate with IBM Rational Software Modeler. The synthesis technique of Schumann [19] was implemented in Java and the author mentioned the intention to integrate the technique into the commercial UML tool Magic Draw. We were, however, unable to find more recent publications by this author since the original publication of the work in 2008 confirming that this actually happened. The synthesis technique of Selonen and colleagues [7], [8, 22] was implemented in an industrial software development environment by the same authors: TED. Maier and Zndorf [21] implemented their synthesis technique in the Fujaba environment, which is a free open source UML CASE tool environment, to provide roundtrip engineering support. Ziadi and colleagues [18] developed a prototype of their synthesis technique in Java. Mäkinen, and Systä [10] implemented their synthesis technique in their own prototype tool.

Kang and colleagues [17] mentioned that they were planning to improve their synthesis technique by automating its manual steps. Khriss and colleagues similarly deferred complete automated support (in their case adding editors for the UML diagram types involved in their synthesis technique) to future work [20]. We were unable to find more recent publications by these authors discussing these aspects since the initial publications (2010 and 1998, respectively).

### 3.6 Summary

Table 4 shows some results regarding the five research questions together. It is interesting to note the absence of synthesis techniques published between 2004 and 2010. The presence of seven primary studies with an old version of UML (70%) [7, 8, 10, 19-22] shows that the issue of the synthesis of UML diagrams started to become relevant from the initial launch of the UML (it has been evolving since the second half of the 1990s [4]). Furthermore the three primary studies that rely on the recent UML versions 2.x [16-18] prove that the new version of the metamodel did have an impact on research in this domain.

Another interesting aspect is that six of the 12 techniques synthesize UML sequence diagrams (50%).

Observations about the UML diagrams that involve synthesis techniques show that the synthesis of state machine diagrams from a collection of scenarios (i.e., sequence diagrams) has received a lot of attention. Another aspect that should be highlighted is represented by the three techniques (25%) that synthesize class diagrams. Finally we have observed that the researcher who is most actively involved in UML synthesis techniques is Petri Selonen [7, 8, 22], with three publications.

**Table 4.** Summary of synthesis techniques and support

| Ref. | Research papers | UML version | Year | Diagram synthesized | Technique used | Automatic support |
|---|---|---|---|---|---|---|
| [16] | PS | 2.1.1 | 2010 | from IOD to SCD | Synthesis Algorithm | YES |
| [8] | ER | 1.3 | 2001 | from SD to CD | Two transformations approaches | YES |
| [22] | PS | 1.3 | 2000 | from SD to CD | Synthesis Algorithm | YES |
| [10] | PS | 1.3 | 2001 | from SD to SCD | Synthesis Algorithm | YES |
| [21] | PS | 1.4 | 2003 | from SD to SCD | Synthesis Algorithm | YES |
| [18] | PS | 2.0 | 2004 | from SD to SCD | Algebraic framework and Synthesis Algorithm | YES |
| [19] | PS | 1.3 | 2000 | from SD to SCD | Synthesis Algorithm | YES |
| [17] | PS | 2.1.2 | 2010 | from SD to AD | Mapping rules and Synthesis Algorithm | NO |
| [7] | VR | 1.4 | 2003 | from SD to SCD<br>from SCD to SD<br>from SCD to CD | Synthesis Algorithm | YES |
| [20] | PS | 1.3 | 1999 | COD to SCD | Synthesis Algorithm | NO |

### 3.7 UML consistency rules extraction

Since the different UML diagrams that are constructed during a specific software development specify different aspects of one, and only one model, these diagrams must be consistent with one another. This is true regardless of whether the diagrams are generated by hand (using a CASE tool for instance) or synthesized from other diagrams. The UML diagram synthesis techniques we have collected therefore enforce some kind of consistency between the input diagrams of the synthesis and the

diagrams being synthesized. This section summarizes the consistency rules we were able to collect from the different synthesis techniques (the primary studies): Table 5.

No one consistency rule was found in the proposals [10,19]. Paper [22] presented the same rules as in [8] but with a new characteristic that can be used to synthesize class diagrams with operation description annotations.

**Table 5.** Summary of the UML consistency rules

| S. T. | UML consistency rules | Ref. |
|---|---|---|
| **SD TO SCD** | 1. A reception in the SD becomes an event in the SCD and emissions become actions. | [18] |
| | 2. For a transition associated with a reception, the action part will be empty, and for transitions associated with actions, the event part will be empty. | |
| | 3. SCDs generated will be sequences of states, and will contain a single junction state that corresponds to the state reached when all events situated on an object lifeline have been executed. | |
| | 4. When an object does not participate in an interaction, the projection of an SD on this object's lifeline is the empty word | |
| | 1. For a single life line within a sequence diagram, incoming messages are interpreted as events attached to certain transitions and outgoing messages are interpreted as do-actions of certain states. | [21] |
| | 2. Following the lifeline of a given object, try to reuse already existing transitions and states in the corresponding SCD for as long as possible. Otherwise new transitions and states are created. | |
| | 3. For each such possible start element, compute the number of subsequent messages that match subsequent SCD elements without the need to create new elements. | |
| | 1. Map items in the message trace in SD to transitions and states in an SCD. | [7] |
| | 2. Messages sent in SD are regarded as primitive actions associated with states in SCD. | |
| | 3. Each message received in SD is mapped onto a transition in SCD. | |
| | 4. A synthesized SCD is deterministic, i.e., there cannot be two similarly labeled leaving transitions in any particular state, unless their guards are mutually exclusive. Two applicable transitions cannot be satisfied simultaneously. | |
| | 5. A completion transition and a labeled transition cannot leave the same state unless their guards differ. | |
| **SD to CD** | 1. Generate a class for each classifier role with a distinctive class name. Transfer stereotypes, such as «actor» and «active» into the respective class. | [8] |
| | 2. For each message, if an association between the base classes of the sending and receiving classifier roles does not exist, generate the corresponding association ends and an association representing the communication connection for the message in question, and link them together. | |
| | 3. For each class receiving a message, if an operation with the same name does not already exist, generate a new operation for the class. Mark the navigability of the association. | |
| | 4. If there are arguments attached to the message, generate new parameters for the operation with corresponding UML basic types and type expressions. Add this operation to the corresponding class if it does not already have an operation with the same name and signature. | |

| | | |
|---|---|---|
| | 5. If the message is marked with a stereotype «signal» and a signal with the same name does not exist, generate a new signal. Associate the corresponding class with this signal. | |
| | 6. If the message is marked with a stereotype «become», the sending and receiving classifier roles are equated (this rule assumes that an object cannot dynamically change its type). | |
| | 7. If the message has an object that appears in the sequence diagram as an argument, we add an association between the corresponding classes of the sending object and argument object, if one does not already exist and the classes are not the same. | |
| | 8. If the message is a return message, containing only a return value of some type, we conclude the return type of the operation of the preceding message. | |
| | 9. The sequence diagram might contain possibly contradictory information, for example an active and a passive object of the same class. In this case, the result of the transformation operation is undefined | |
| **IOD to SCD** | 1. Synthesis for Normal Edges: normal edges in an EIOD are interpreted as a weak sequential composition but taking into account flow final and final nodes. | [16] |
| | 2. Synthesis for Parallel Fork/Join Edges: parallel fork/join edges lead to orthogonal regions in the HFSMs generated. Since fork/join edges are well nested, there is no ambiguity as regards deciding where the orthogonal regions should be placed. | |
| | 3. Synthesis for Preempting and Suspending Edges: for preempting and suspending edges, composite states are introduced into the HFSMs generated. | |
| | 4. Synthesis for Negative Edges: negative edges result in orthogonal regions in the HFSMs generated. Negation is handled by monitoring for the negative events and transitioning to a special error state if they occur. The negative events being monitored are placed in an orthogonal region so that, if the sequence of negative events ever occurs (even with other events interleaved), then the error state will be entered. | |
| | 5. Synthesis for Multiple Concurrent Nodes: if a node group is marked as having multiple concurrent nodes (i.e., it has an asterisk attached to it), then the EIODs semantics state that the node group can be replaced with a parallel fork/join edge in which the node group is repeated an undetermined number of times. | |
| **SD to AD** | 1. A self call action of an object in a sequence diagram translates to an action of the corresponding participant of an Activity Diagram. | [17] |
| | 2. A message passing description in a Sequence Diagram can be translated using the notations such as send signal, receive signal, object nodes, and transition between the signal node and the object node. | |
| | 3. When a reference is used in a Sequence Diagram, we omit explicit reference notation and simply use a reference name. This should then be later expanded after the expansion of the reference of a Sequence Diagram is recursively translated into the Activity Diagram and connected to the rest of the Activity Diagram | |
| | 4. When references are used, the reference type of a Sequence Diagram specification is translated into the interaction occurrence type of an Activity Diagram. | |
| | 5. The application of the alt operator of a Sequence Diagram can be transformed into an Activity Diagram by using the synchronization bars, decision nodes and the merge nodes. | |

| | | |
|---|---|---|
| | 6. In order to transform a loop operator of a Sequence Diagram into an Activity Diagram, setup, test, and body sections should be identified first. After finishing the body section, each process that participates in the body section goes back to the join node to check the condition again. | |
| | 7. The par operator of a Sequence Diagram represents the parallel composition of multiple scenarios. By using the fork and join nodes of an Activity Diagram, a participant can be split into multiple threads and joined together. | |
| **COD to SCD** | 1. Create an SCD for every distinct class implied by the objects in the COD. | [20] |
| | 2. Introduce as state variables all variables which are not attributes of the objects of COD. | |
| | 3. Create transitions for the objects from which messages are sent. | |
| | 4. Create transitions for the objects to which messages are sent. | |
| | 5. For all SCDs, put the set of transitions generated into correct sequences, connecting them by states, split bars and merge bars. | |
| **SCD to SD** | 1. Actions are messages sent or actions performed by the object whose behavior is described by the SCD. Event triggers of transitions are messages received by the object. | [7] |
| | 2. Actions that are attached to transitions or states are transformed into corresponding actions of an interaction. These actions imply messages and classifier roles. | |
| | 3. Signal events and call events associated with transitions or states are also mapped onto send actions and call actions, respectively. | |
| | 4. The external stimuli can also be given as an SCD, thus defining the response of an external actor to the output of the system. | |
| **SCD to CD** | 1. Signal events of transitions and states are mapped onto signals and ultimately onto behavioral features that define their context. | [7] |
| | 2. Call events are mapped onto operations of classifiers. | |
| | 3. Send actions and call actions are mapped onto signals and behavioral features, and operations, respectively. | |
| | 4. Actions are also used to infer stimuli and corresponding links which are in turn mapped onto associations and association ends of classifiers. | |
| | 5. The context of the state machine is mapped onto the classifier for which the behavior is defined. | |

## 4 Threats to Validity

The main threats to the validity of an SMS like ours are related to publication bias, selection bias, inaccuracy in data extraction, and misclassification [28].

As it is impossible to completely cover every publication written on our topic, we acknowledge that some relevant papers might not have been included. We used five search engines to collect journals, conferences and workshops proceedings that are relevant to UML synthesis techniques; we did not consider grey literature (e.g., PhD theses, books) or unpublished results (e.g., technical reports) because they might have affected the validity of our results.

Selection bias refers to the distortion of a statistical analysis owing to the criteria used to select publications. We attempted to solve this problem by defining our inclusion and exclusion criteria in order to gather the most relevant papers regarding UML diagrams synthesis techniques. To help ensure an unbiased selection process, we de-

fined research questions in advance, organized the selection of articles and finally created and followed a multi-phase process to execute the SMS. We would also like to mention that during the data extraction process, there was the possibility of subjectivity when we decided what was (and what was not) related to our topic. This interpretation might have affected the results. The data was extracted from the papers by one researcher (the first author of the paper).

Finally, when considering the limited number of primary studies in this SMS, we know that the coverage of this SMS might be the most inherent threat to validity, even though we followed rigorous guidelines that are well-known in the empirical software engineering community [12]. The classification scheme that we provide in this paper may be used as a starting point by future researchers.

## 5 Conclusion

This work presents the results obtained after applying the Systematic Mapping Study (SMS) protocol whose aim was to identify and evaluate the current approaches for the synthesis of UML diagrams from other UML diagrams. No such mapping study or review existed prior to our work.

The SMS was carried out by following well-known guidelines [12]. A significant amount of space in this paper has been used to discuss how these guidelines were followed, since we felt that it was of the upmost importance to allow readers to precisely understand the results and conclusions, and to allow others to replicate our study or compare our results with others.

From an initial set of 67 papers published in literature and extracted from five scientific research databases, a total of ten were eventually analyzed in depth, by following a precise selection protocol driven by five research questions. We observe that (in no particular order of importance):

- There is no commercially available UML CASE tool standard with which to run UML synthesis techniques between UML diagrams;
- The most frequently used UML diagram type as a source for synthesis is the sequence diagram (66.7%), and the most synthesized diagram types are the state machine diagram (58.3%) and the class diagram (25%). This is not entirely surprising since these are the most frequently used UML diagrams [25].
- We did not find any synthesis techniques for Communication, Protocol State Machine, Object, Interaction, Composite Structure, Timing and Deployment Diagrams. This may either mean that no such synthesis really makes sense, or that additional research on synthesis techniques involving all 14 UML diagrams is warranted.
- Synthesizing UML diagrams from other UML diagrams is not a very active line of research since the most recent papers were published in 2010 [16, 17] and we only collected ten primary studies.

Nevertheless, considering that any UML synthesis technique needs to enforce consistency between the input UML diagram and the output synthesized UML diagram (

section 3.7), this topic is a very mature and current topic [6]. Indeed, thanks to the synthesis techniques presented in the papers collected, a UML model, and its UML diagrams attain a higher quality: increased consistency and correctness, because they are either produced or updated automatically, or are checked against each other using the synthesis techniques. We feel that the time is ripe for these UML synthesis techniques to be put through their paces and that in-depth studies on how they can most effectively support UML consistency rules should now be undertaken.

These observations definitely call for future work. Most importantly, additional work should be planned to address the threats to validity we have mentioned earlier in this paper.

## Acknowledgements

This research has been partly funded by a Discovery grant of the Natural Sciences and Engineering Research Council of Canada and the GEODAS-BC project (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01).

## References

1. Mukerji, J., Miller, J.: Overview and guide to OMG's architecture. Object Management Group (2003), http://www.omg.org/mda/
2. Thomas, D.: MDA: Revenge of the modelers or UML utopia? IEEE Software 21, 15–17 (2004)
3. Genero, M., Fernández-Saez, A.M., Nelson, H.J., Poels, G., Piattini, M.: A Systematic Literature Review on the Quality of UML Models. Journal of Database Management 22, 46-70 (2011)
4. OMG: OMG Unified Modeling LanguageTM - Superstructure Version 2.4.1. Object Management Group (2011)
5. Pender, T.: UML Bible (1st ed.). John Wiley & Sons, Inc., New York, NY, USA (2003)
6. Torre, D., Labiche, Y., M., G.: UML consistency rules: a systematic mapping study. 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014). ACM, London, UK (2014)
7. Selonen, P., Koskimies, K., Sakkinen, M.: Transformations between uml diagrams. Journal of Database Management 14, 37-55 (2003)
8. Selonen, P., Koskimies, K., Sakkinen, M.: How to Make Apples from Oranges in UML. 34th Annual Hawaii International Conference on System Sciences ( HICSS '01), vol. 3. IEEE Computer Society, Washington, DC, USA (2001)
9. Manna, Z., Waldinger, R.J.: Toward automatic program synthesis. Commun. ACM 14, 151-165 (1971)
10. Mäkinen, E., Systä, T.: MAS — an interactive synthesizer to support behavioral modelling in UML. 23rd International Conference on Software Engineering

(ICSE '01). IEEE Computer Society, Washington, DC, USA, Washington, DC, USA (2001)
11. Arksey, H., O'Malley, L.: Scoping studies: towards a methodological framework. International Journal of Social Research Methodology 8, (2005)
12. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Keele University (2007)
13. Budgen, D., Turner, M., Brereton, P., Kitchenham, B.: Using mapping studies in software engineering. Psychology of Programming Interest Group Workshop, pp. 195–204, Lancaster University (2008)
14. Wieringa, R., Maiden, N., Mead, N., Rolland, C.: Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requirements Eng. 11, 102-107 (2005)
15. Brereton, P., Kitchenham, B.A., Budgen, D., Mark Turner, Khalil, M.: Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Softw. 80, (2007)
16. Whittle, J., Jayaraman, P.K.: Synthesizing hierarchical state machines from expressive scenario descriptions. ACM Trans. Softw. Eng. Methodol. 19, (2010)
17. Kang, S., Kim, H., Baik, J., Choi, H., Keum, C.: Transformation Rules for Synthesis of UML Activity Diagram from Scenario-Based Specification IEEE 34th Annual Computer Software and Applications Conference (COMPSAC '10), pp. 431-436. IEEE Computer Society, Washington, DC, USA (2010)
18. Ziadi, T., Helouet, L., Jezequel, J.-M.: Revisiting Statechart Synthesis with an Algebraic Approach 26th International Conference on Software Engineering (ICSE '04). IEEE Computer Society (2004)
19. Schumann, J.: Automatic debugging support for uml designs. In: Ducasse, M. (ed.) Fourth International Workshop on Automated Debugging, (2000)
20. Khriss, I., Elkoutbi, M., Keller, R.K.: Automating the Synthesis of UML StateChart Diagrams from Multiple Collaboration Diagrams. In: Muller, J.B.a.P.-A. (ed.) First International Workshop on The Unified Modeling Language: Beyond the Notation (UML '98), pp. 132-147. Springer-Verlag, London, UK, UK (1998)
21. Maier, T., Zndorf, A.: The Fujaba Statechart Synthesis Approach. Workshop on Scenarios and State Machines (ICSE '03), Portland, Oregon, USA (2003)
22. Selonen, P., Systä, T.: Scenario-based Synthesis of Annotated Class Diagrams in UML. Workshop: Scenario-based round-trip engineering (OOPSLA '00) vol. 20, pp. 26-31, Tampere University of Technology, Software Systems Laboratory (2000)
23. Mäkinen, E., Systä, T.: An Interactive Approach for Synthesizing UML Statechart Diagrams from Sequence Diagrams. OOPSLA 2000 Workshop: Scenario-based round-trip engineering, pp. 7-12 (2000)
24. Makinen, E., Systa, T.: Implementing minimally adequate synthesizer. Technical Report, Dept. of Computer and Information Sciences, University of Tampere (2000)
25. Dobing, B., Parsons, J.: How UML is used. ACM 49, 109-113 (2006)

26. Whittle, J., Schumann, J.: Generating statechart designs from scenarios. 22nd international conference on Software engineering (ICSE '00). ACM (2000 )
27. Systä, T.: Incremental construction of dynamic models for object-oriented software systems. Journal of Object-Oriented Programming 13, 18-27 (2000)
28. Sjoberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.-K., Rekdal, A.C.: A Survey of Controlled Experiments in Software Engineering. IEEE Trans. Softw. Eng. 31, 733-753 (2005)